POLYTECH NICE SOPHIA | UNIVERSITÉ CÔTE D'AZUR

# Rapport de projet

# Programmation web

**Client Side**

M. François Michaudon & M. Hugo Mallet

**Rédigé par :**
Djotiham NABAGOU (Membre de l'équipe ADKN)

# PLAN


Identifiant github 3

Tâches effectuées 3

Stratégie employée pour la gestion des versions avec Git 3

Solutions choisies 4

Difficultés rencontrées 4

Temps de développement / tâche 5

Code 5

## 1. Identifiant github

Mon identifiant github est le suivant : **djotihamNABAGOU**

## 2. Tâches effectuées

Au cours de ce projet, j'ai principalement travaillé sur la **map** afin de permettre une visualisation facilement compréhensible du niveau de l'épidémie dans chaque département. Toutefois, cette fonctionnalité m'a pris plus de temps que prévu en raison de la complexité de son implémentation, et ceci reste au final le seul composant que j'ai implémenté dans ce projet.

Outre la map, j'ai rédigé et mis en forme le **readme** exigé pour le rendu du projet le rendu.

## 3. Stratégie employée pour la gestion des versions avec Git

Ayant l'habitude d'utiliser Git pour nos différents projets, nous avons employé le système habituel :

- Le branching

La branche "**main**" est la branche principale du projet. Une fois le projet initialisé, notre réflexe est de partir sur une branche "**develop**" sur laquelle nous développons et mettons en commun toutes nos fonctionnalités. La branche principale et éventuellement d'autres branches dont on aurait besoin d'effectuer le suivi sont contrôlées par une CI (Github Actions) qui se charge de lancer un workflow et éventuellement déployer le projet souhaité.

Dans notre cas, nous avions décidé de déployer uniquement le backend au départ sur un serveur ssh distant (OVH) afin de centraliser les données du backend et d'accéder aux mêmes informations. Ceci était surtout possible car un membre de l'équipe (Paul KOFFI) disposait de ce serveur et a bien voulu uploader le projet dessus pour le groupe.

Quant aux autres branches, nous nous inspirons de ce qui se fait dans bon nombre d'entreprises : le pattern branching de Github. Lorsque qu'un membre de l'équipe estime qu'il a une fonctionnalité à développer qui peut impacter sur le travail des autres ou qu'il a besoin de continuer à travailler avec la version actuelle du projet sans nouvelles modifications futures ou encore quand il estime que c'est plus propre pour lui et pour le projet, il crée alors une branche dédiée à sa fonctionnalité. Une fois la fonctionnalité développée, il merge ses modifications sur la branche de développement et s'assure de bien intégrer ses modifications avec la version en cours puis il supprime sa branche.

Ceci nous permet d'être bien organisés dans nos commits et de retracer facilement tout bug récent. Les branches de fonctionnalité n'ont pas forcément de CI qui leur est associée car elles seront contrôlées lors de leur fusion avec la branche de développement commune.

Une fois toutes les modifications intégrées et validées sur la branche commune de dev, elles sont poussées sur la branche principale "**main**" qui statue d'une version stable du projet à l'instant t. une release est ensuite créée selon un tag passé en paramètre afin de figer la version actuelle et continuer à ajouter d'autres features sur la branche principale.

- Les tickets

Le gestionnaire de tickets de Github nous permet de documenter nos commits et ainsi retracer la fonctionnalité que fait chaque partie de code au moment où elle est poussée sur github. Ainsi, nous référençons chaque commit à un ticket spécifique. Aussi ce gestionnaire nous permet d'assigner des tâches et effectuer des commentaires sur les commits des autres membres lors de leurs pull requests.

## 4. Solutions choisies

Pour l'implémentation de la map, j'ai choisi **Leaflet** après avoir effectué des recherches sur les différentes possibilités de représenter une carte en react. Leaflet ressortait très souvent des recherches et disposait facilement d'aide en ligne et d'une communauté active pour les éventuels bugs. C'est la première carte que j'ai testé et je n'ai pas eu à en choisir une autre notamment parce qu'elle m'a permis d'intégrer facilement un layer (type de représentation de la carte, exemple : représentation par vision satellite, représentation géographique, démographique, 3D...) adéquat et libre d'utilisation. En effet le layer utilisé pour réaliser la map est le layer **OpenStreetMap.Mapnik** d'OpenStreetMap. Il permet d'avoir une cartographie semblable à celle par défaut de Google Map qui dans le cas de la France, nous permet d'avoir un découpage en départements, qui nous est nécessaire pour la visualisation choisie.
Une alternative aurait été d'utiliser **Mapbox** qui permet aussi d'implémenter des cartes mais en comparaison à OpenStreetMap, Mapbox nécessite un token d'utilisation qu'on obtient uniquement après avoir créé son compte puis construire son propre layer.

## 5. Difficultés rencontrées

Une fois la carte mise en place, l'objectif visé était de faire un zoom sur la France afin d'y montrer les différents départements touchés par la pandémie et visualiser grâce à un indicateur (couleur) les départements allant des moins touchés aux plus touchés. En cliquant sur un département, son nom et le nombre de cas covid doivent apparaître dans une info bulle.

Les principales difficultés consistaient à trouver le bon découpage de carte pour y afficher les données et passer en paramètre les données à la carte de telle façon que la carte soit actualisée en fonction du filtre appliqué.

Les contours de départements se faisant grâce à des des points geojson positionnés sur la map, il fallait trouver la ressource permettant de faire ce découpage. Après plus essais, le site https://france-geojson.gregoiredavid.fr/ m'a permis d'extraire un découpage de la france selon tous ses départements et a permi la suite des fonctionnalités

Le deuxième point était un réel point de blocage pendant un long moment car il fallait transiter les données sur la carte en prévoyant de les actualiser lors de l'application du filtre. L'utilisation du **useEffect** pour observer la nouvelle valeur du filtre et mettre à jour la carte ne donnait pas de résultat. Il a fallu passer par un système de clé / valeur qui affectait une nouvelle valeur unique au composant carte à chaque fois que le filtre était appliqué, ce qui a permis d'actualiser la carte

## 6. Temps de développement / tâche

En raison de la complexité du composant carte, son implémentation m'a pris 3 jours.

## 7. Code

- Exemple de code propre

https://github.com/wak-nda/prog-web-ADKN/blob/lab/frontend/src/components/MapComponent.js

Ce composant met en lumière la bonne utilisation des hooks **useEffect** et **useState** qui couplé au système de clé / valeur attribué à la carte, ont permis de pouvoir actualiser les données suite à l'application d'un filtre.

- Exemple de code à améliorer

https://github.com/wak-nda/prog-web-ADKN/blob/lab/frontend/src/entities/LegendItems.js

Cette classe moque les données de légende à afficher en bas de la carte pour une meilleure compréhension. Le problème est que les données renvoyées à la carte ne sont pas figées et peuvent changer en fonction d'un filtre sélectionné. il est donc nécessaire ici de passer en paramètre les données de la carte telles que les chiffres statistiques, la couleur et le texte d'affichage afin que le composant carte soit bien mis à jour.